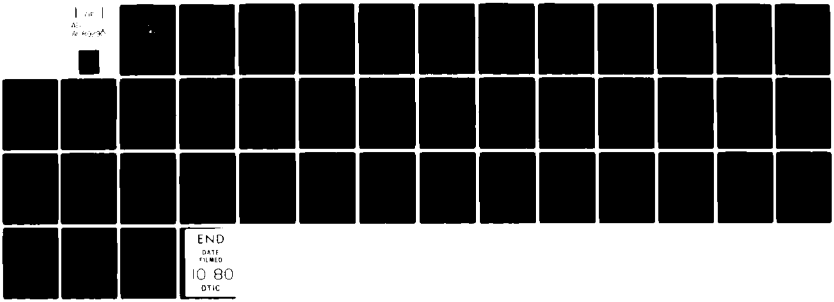
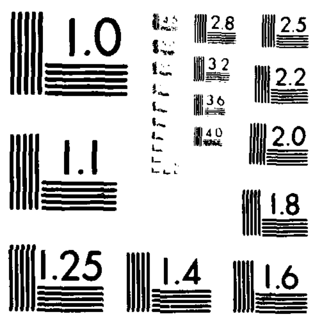
AD-A089 290 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 15/7
THE STAR FIELD MODULE.(U)
JUN 80 J K HARTMAN
UNCLASSIFIED NPS55-80-023 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

NPS55-80-023

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD A089290

DTIC
ELECTE
SEP 19 1980

9 Technical rept.

THE STAR FIELD MODULE

by

James K. HARTMAN

Jun[redacted]80

43

Approved for public release; distribution unlimited.

Prepared for:
The U.S. Army Training & Doctrine Command
Fort Monroe, VA.

DDC FILE COPY

80 9 18 014

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

Jack R. Borsting
Provost

The work reported herein was accomplished with the support of the U. S. Army Training & Doctrine Command, Fort Monroe, VA.

Prepared by:

James K Hartman

James K. Hartman
Department of Operations Research

Reviewed by:

C Howard, acting

Michael G. Sovereign, Chairman
Department of Operations Research

Released by:

William M. Tolles

William M. Tolles
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS55-80-023	2. GOVT ACCESSION NO. AD-A087 290	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The STAR Field Module		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James K. Hartman		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS MIPR No. CD 1-80
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army TRADOC Fort Monroe, VA 23651		12. REPORT DATE June 1980
		13. NUMBER OF PAGES 43
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

STAR
Combat model
Simulation
Minefield
Obstacle

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the FIELD module of the STAR combat simulation. The FIELD module is used to simulate minefields, obstacles, and other terrain related battlefield features. The FIELD routines are described and typical uses are discussed.

TABLE OF CONTENTS

A

I. The Field Concept

A. Introduction

This report describes a portion of the STAR (Simulation of Tactical Alternative Responses) combined arms ground/air combat simulation model. STAR has been developed by students and faculty of the Operations Research Department at the Naval Postgraduate School as a tool for investigating ground/air combat. The model is programmed in SIMSCRIPT II.5, and a knowledge of that language is presumed in this report. Documentation of various other aspects of the model is given in references [1] - [5].

The purpose of the present report is to describe the Field Module of STAR. The Field Module is responsible for simulating vehicles encountering minefields and other similar battlefield features. The remainder of this chapter will introduce the general concept of a field and the two types of actions related to fields. Chapter II will describe the data structures representing fields and the basic subroutines for creating fields and monitoring vehicle location with respect fo fields. Chapter III will discuss the modifications required in the existing STAR model to incorporate the Field Module. Finally, Chapter IV will consider the actions - the tactical responses - that occur as a result of fields.

B. Fields

Definition: A <u>field</u> is an area on the battlefield which influences battle actions and hence battle outcome.

Examples of fields include the following sorts of areas on the battlefield - some manmade and others naturally occurring:

Minefields - dug in or scatterable
Engineered or natural obstacles
River crossings
Bridges
Towns
Forests
Artillery trigger areas
Residual nuclear or chemical effects areas.

Forests are currently handled separately in the LOS module [2]. The Field Module will be set up to be able to handle the remaining fields.

Fields are represented in the STAR model as geometric areas overlaid on the terrain. The field module has the following functions:

1. For each element in the simulation, monitor when the element enters/exits each field.
2. When an element enters a field, perform designated actions (e.g. lower mine plow, reduce speed, change formation).
3. When an element leaves a field, restore its condition to the normal state.
4. Inside the field, perform designated actions (e.g. detonate mines).
5. Create fields at the beginning of the simulation and, if required, during the execution of the simulation.
6. Destroy fields during the simulation if required.

Since the Brigade level STAR model has the potential for modelling over a thousand elements, the field monitoring routines must be organized efficiently to avoid excessive overhead.

C. Field Actions

Field actions are things that happen because vehicles encounter fields. Typical actions which can be simulated include the following:

1. Change speed.
2. Change movement formation.
3. Stop for a given time, then continue.
4. Go into a hasty defensive posture.
5. Detonate a mine.
6. Lower or raise mine plow.
7. Call for artillery or air strike.
8. Mount or dismount infantry.
9. Change movement path in attempt to bypass field.
10. Attempt to clear obstacles or mines.
11. Attempt river crossing.
12. Respond to nuclear effects such as radiation, fires, blowdown.

The field actions which occur for a given field will typically depend on the field type (e.g. minefield), on parameters of the field (e.g. mine density), and on the tactical situation (e.g. under fire?). In this section we discuss the general concept of field actions and how they are triggered. Implementation of specific actions will be covered in Chapter IV.

There are two distinct kinds of field actions. Most field actions will occur immediately when vehicles cross the field boundaries (entering or exiting.) These actions, which we will call boundary actions, are easily triggered at the instant of entry or exit if the movement model can detect field boundary crossings. Other actions, however, occur somewhere inside the field. A mine detonation is the most obvious example. These actions are called internal actions. They are initiated by a field entry, but do not occur until the vehicle has moved into the field. To trigger

these actions, the simulation must measure the distance moved since field entry.

It should be noted that neither type of field action can be conveniently modelled by scheduling SIMSCRIPT events. This is because a scheduled event occurs after a designated time interval has passed whereas field actions occur at designated locations. Since vehicles in STAR need not move at constant (or easily predictable) speed, it is usually impossible to predict when in time a field action will occur. Instead, the field module must monitor the location of each simulation element with respect to each field at every time of interest in the simulation. This is a task involving a potentially huge amount of computation.

One approach to monitoring location with respect to fields would use the movement model. For each element, at each move increment, the procedure would loop over all fields, testing whether the element was in the field. If the in/out status changed from a previous move increment, then a boundary action would be performed. This approach was considered and rejected for several reasons:

1. We suspect that it would be quite expensive to compute.
2. The approach as described only locates field boundaries to the accuracy of the current move increment's length. (This could be overcome with additional computation).
3. Internal actions cannot be easily handled using this approach.

An alternative computational scheme is implemented in the current field module. For each element we compute and store the distance to the nearest field boundary action (entry, or exit) in the element attribute FLD.BDY.DIST(TANK). Similarly, we store the distance (if any) to a pending

internal action in the attribute FLD.INT.DIST(TANK). Then for each move increment these distances are decremented. When the remaining distance goes to zero, the appropriate action is triggered and the distance to the next action is computed. The main advantage of this process is that we need to loop over all fields only when a new distance is computed, and this will only be necessary when

1. The vehicle's direction of movement changes.
2. The vehicle encounters a field action.
3. New fields are created or old ones destroyed.

It also allows increased accuracy and handles boundary actions and internal actions using the same mechanism.

Details of the computation are reserved for Chapter II.

D. Overlapping Fields

A major problem which must be considered in planning the Field module is the problem of overlapping fields. If an element can be in several fields simultaneously, ambiguities can arise as to the intended field actions. The geometry of the field module can recognize multiple field entries and exits in whatever order they occur without any problems. However, making the action routines smart enough to take combinations of fields into account is probably prohibitively complex.

Unless special circumstances dictate otherwise, we plan to make the field action routines for a given field ignorant of whether the element is in other fields. Thus an element can safely be in two (or more) fields at one time as long as those fields influence different aspects of the element's behavior. If, however, field 1 increases an element's speed by 50% and field 2 decreases speed by 2 m./sec, then the effect of being in

both fields will be ambiguous. If the initial speed is 5 m/sec, then the resulting speed will be 5.5 m/sec if field 1 is entered first, or 4.5 m/sec if field 2 is entered first. The user should be aware of such possible ambiguities and avoid them by careful definition of the fields required for a given scenario.

Overlapping internal actions are also limited by the computational procedure of the Field module. Since an internal action is initiated (planned) when the field is entered, but doesn't actually occur until a designated distance into the field, we need to store this distance for each element (as an attribute called FLD.INT.DIST). Since only one such internal distance is stored for each element, the model will not correctly simulate an element which is simultaneously in two different fields which both involve internal actions.

These limitations are part of the current coding of the Field module. They doubtless could be overcome by more involved logic at an increased cost in coding effort, computer storage, execution time, and model clarity.

II. The Field Geometry Module for STAR

A. Overview of the Module

This chapter discusses the details of the geometric aspects of the field module. These include the definition and creation of fields and the process of monitoring field entry and exit. The Field module involves a new class of SIMSCRIPT temporary entities (FIELDS), along with several new subroutines for processing fields, which are discussed in this chapter. In addition, several existing routines must be modified to interface with the new module. (Chapter III).

B. The Temporary Entity - FIELD

In the STAR combat simulation, fields are modelled as elliptical overlays on the terrain. For purposes of field planning and model input data the shape of each field ellipse is described by the following parameters (see Fig. 1.):

```
XC.FLD \  X and Y battlefield coordinates of the
YC.FLD /  center of the ellipse

SMAJ.FLD   semi-major axis length
SMIN.FLD   semi-minor axis length

ANG.FLD    orientation angle in radians measured
           counterclockwise from east to the major axis.
```

For computational purposes the boundary of the field is described by the quadratic equation

```
PXX.FLD*(X-XC.FLD)**2 + PYY.FLD*(Y-YC.FLD)**2

     + PXY.FLD*(X-XC.FLD)*(Y-YC.FLD) = 1.0
```

Where the quadratic coefficients are defined as

```
    CANG = cos(ANG.FLD)
    SANG = sin(ANG.FLD)
 PXX.FLD = (CANG/SMAJ.FLD)**2 + (SANG/SMIN.FLD)**2
 PYY.FLD = (SANG/SMAJ.FLD)**2 + (CANG/SMIN.FLD)**2
 PXY.FLD = 2*SANG*CANG*(1.0/SMAJ.FLD**2 - 1.0/SMIN.FLD**2)
```

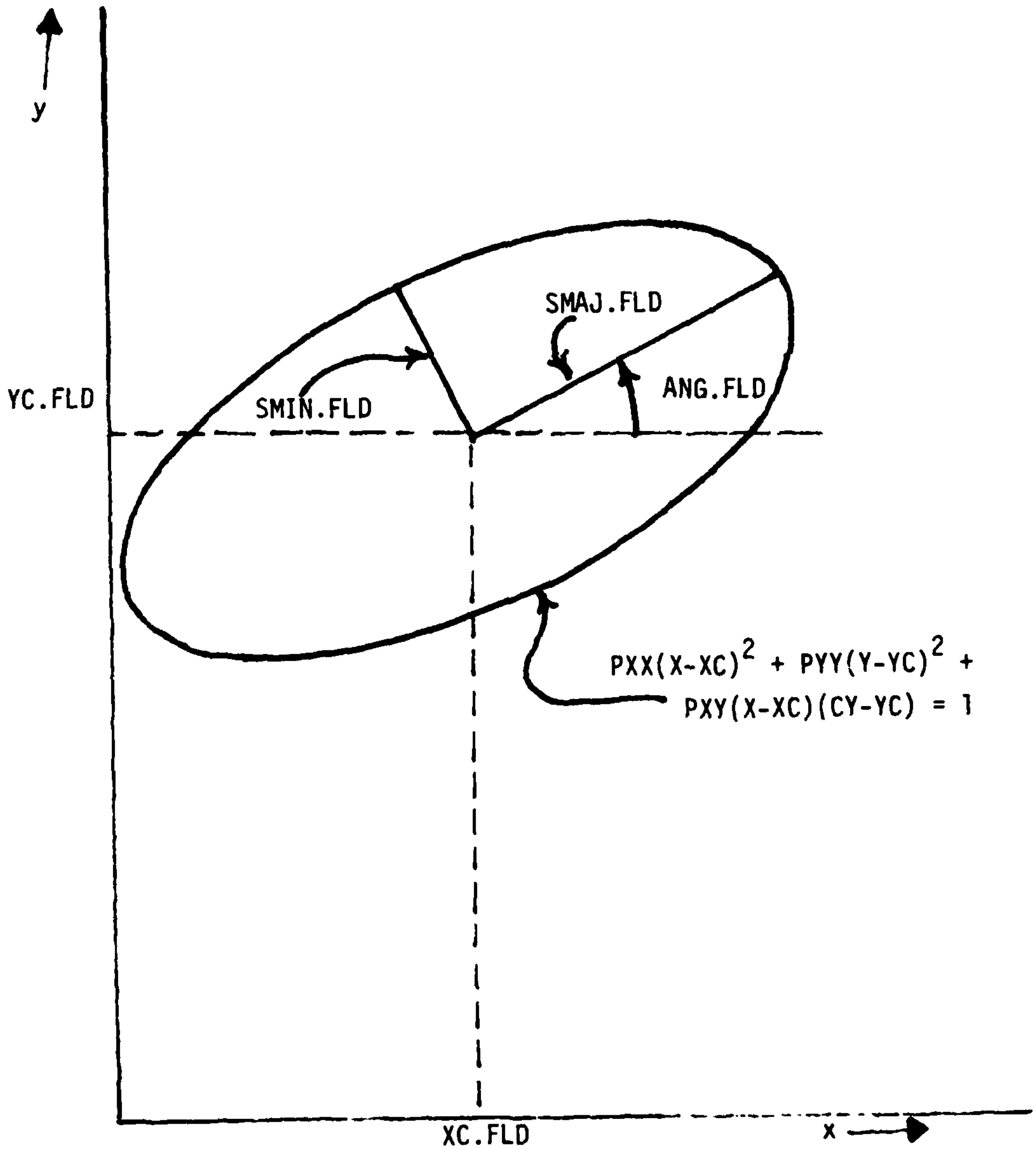


Figure 1. Field Shape Parameters

Each field is a SIMSCRIPT temporary entity in the entity class FIELD. The computational field parameters are stored as entity attributes as follows:

XC.FLD(FIELD) YC.FLD(FIELD) PXX.FLD(FIELD) PYY.FLD(FIELD) PXY.FLD(FIELD)	Defined as above - all are real variables

In addition, each field has several other attributes:

NAM.FLD	Sequential ID number in order of creation (Integer)
TYP.FLD	Field type code (Integer)
P1.FLD P2.FLD P3.FLD P4.FLD	Real variables defining other field parameters (e.g. minefield density) which influence field action routines. May have different meanings for different field types.

The distinction between the field type and the P1-P4 parameters is one of convenience. If two fields (say a scatterable minefield and a dug-in minefield) will both use the same field action routines, then they should be given the same field type, and the parameters should be used to distinguish between them. If, however, the actions triggered by one field are of a different sort than those resulting from the other field, then different field type codes should be used to describe them. We plan to have separate dedicated routines to trigger the actions for each separate field type. Each of these routines can use any or all of the P1-P4 parameters. Four such parameters are initially provided for; ultimately this number may change.

C. The FLD.SET

For convenience in looping over the currently active fields, a system-owned set called the FLD.SET has been declared. Each field entity is

filed in the FLD.SET as the field is created. For as long as the field remains in the FLD.SET, the simulation will consider it to be an active field and will process it in all field module computations. If a field is removed from the FLD.SET, then the field module will ignore it.

D. Routine FLD.CREATE

Field entities are created by the FLD.CREATE routine. In a typical STAR run, this routine will be called several times by the FLD.INIT routine before the start of the simulation to initialize all fields that are in place at the start of the battle. Then during the simulation, FLD.CREATE may be called at any time to create dynamically emplaced fields (e.g. artillery scatterable mines). The routine assigns the next unused sequence number to the NAM.FLD attribute of the new field and returns this value so that, if desired, the calling program can refer to this specific field in the future.

Given Arguments

XC	(real)	X coordinate of ellipse center
YC	(real)	Y coordinate of ellipse center
SAMAJ	(real)	semi-major axis length
SAMIN	(real)	semi-minor axis length
ANGLE	(real)	proper angle in radians from east to major axis
TYPE	(integer)	field type code
P1	(real)	field parameters
P2	(real)	
P3	(real)	
P4	(real)	

Yielding Argument

NAME	(integer)	sequential ID number of this field

Local Variables

SANG	(real)	sin(ANGLE)
CANG	(real)	cos(ANGLE)

Global Variables

FLDS.CREATED	(integer)	counter for number of fields created so far

FIELD Entity Attributes

XC.FLD	See section B of the chapter for definitions - all are set by this routine for the newly created field.
YC.FLD	
PXX.FLD	
PYY.FLD	
PXY.FLD	
NAM.FLD	
TYP.FLD	
P1.FLD	
P2.FLD	
P3.FLD	
P4.FLD	

Routines Called

None

Events Scheduled

None

Code See Figure 2.

Brief Description

Line 7	Creates the new field entity
Lines 8-14	Set field attributes directly from the input arguments
Lines 15-17	Assign the sequence number
Lines 18-22	Compute the quadratic coefficients
Line 23	Files the field in the FLD.SET

Use Enter with descriptive geometric field parameters.

On exit the new field is created, and its computational parameters are set.

```
ROUTINE FLD.CREATE
'' CREATE ONE FIELD AND SET ITS ATTRIBUTES
    GIVEN XC,YC,SAMAJ,SAMIN,ANGLE,TYPE,P1,P2,P3,P4
    YIELDING NAME
    NORMALLY MODE IS REAL
    DEFINE TYPE, NAME AS INTEGER VARIABLES
    CREATE A FIELD
    LET XC.FLD(FIELD) = XC
    LET YC.FLD(FIELD) = YC
    LET TYP.FLD(FIELD) = TYPE
    LET P1.FLD(FIELD) = P1
    LET P2.FLD(FIELD) = P2
    LET P3.FLD(FIELD) = P3
    LET P4.FLD(FIELD) = P4
    ADD 1 TO FLDS.CREATED
    LET NAM.FLD(FIELD) = FLDS.CREATED
    LET NAME = FLDS.CREATED
    LET SANG = SIN.F(ANGLE)
    LET CANG = COS.F(ANGLE)
    LET PXX.FLD(FIELD) = (CANG/SAMAJ)**2 + (SANG/SAMIN)**2
    LET PYY.FLD(FIELD) = (SANG/SAMAJ)**2 + (CANG/SAMIN)**2
    LET PXY.FLD(FIELD) = 2*SANG*CANG*(1/SAMAJ**2 - 1/SAMIN**2)
    FILE THIS FIELD IN THE FLD.SET
    RETURN
END
```

Figure 2. FLD.CREATE Routine

E. Routine FLD.INIT

The FLD.INIT routine is called once by MAIN before the start of simulation. It reads cards describing the fields to be created before the simulation starts and calls FLD.CREATE for each such field. The FLD.INIT routine is then released from MAIN. The code in Figure 3 is self explanatory with the possible exception of line 9 which converts the angle from degrees (on the input card) to radians for the simulation.

F. Routine FLD.DIST

The FLD.DIST routine computes the distance to the nearest field boundary along an element's direction of movement. The resulting distance is stored in the element's FLD.BDY.DIST attribute. The distance is computed by solving the quadratic equation for the intersection of the ellipse boundary with the movement ray as folows:

Let XCUR,YCUR be the current element location and DX,DY the components of a normalized direction vector for the element,

so DX = cos(element angle of movement)

and DY = sin(element angle of movement).

Then

$$X = XCUR + S*DX \qquad (1)$$
$$Y = YCUR + S*DX$$

give a parametric form of the movement path where the parameter S>0 represents distance moved.

The ellipse boundary is given by

$$PXX(X-XC)^2 + PYY(Y-YC)^2 + PXY(X-XC)(Y-YC) = 1.0, \qquad (2)$$

so plugging (1) into (2) gives a quadratic equation in S. If the quadratic has no real roots, then the movement path does not intersect the ellipse. If the roots are S1 = S2, then the solution is a tangency

```
ROUTINE FLD.INIT
'' CREATE ALL FIELDS THAT ARE IN PLACE AT THE START OF THE BATTLE
    NORMALLY MODE IS REAL
    DEFINE TYPE, NUM, I, NAME AS INTEGER VARIABLES
    USE UNIT 5 FOR INPUT
    READ NUM
    FOR I = 1 TO NUM DO
         READ XC,YC,SAMAJ,SAMIN,ANGLE,TYPE,P1,P2,P3,P4
         LET ANGLE = ANGLE/RADIAN.C
         CALL FLD.CREATE GIVEN XC,YC,SAMAJ,SAMIN,ANGLE,TYPE,P1,P2,P3,P4
              YIELDING NAME
    LOOP
    RETURN
END
```

Figure 3. FLD.INIT Routine

and we ignore it. Otherwise both roots exist and S1 < S2. Negative roots correspond to boundaries already crossed, while positive roots correspond to boundaries which lie ahead. The routine loops over all fields and computes the smallest positive root.

Given Arguments

VEH	(Integer)	pointer to the element being moved

Local Variables

D	(double)	the smallest distance found so far
DX	(double)	cos of element movement direction
DY	(double)	sin of element movement direction
XCUR	(double)	element's current location
YCUR	(double)	
PXX	(double)	field boundary coefficients
PYY	(double)	
PXY	(double)	
A	(double)	coefficients of the quadratic equation $AS^2 + BS + C = 0$
B	(double)	
C	(double)	
S1	(double)	roots of the quadratic (if they exist) with S1 < S2
S2	(double)	

Element Entity Attributes

X.CURRENT	(real)	element's current location and angle of movement
Y.CURRENT	(real)	
DIR.OF.MVMT	(real)	
FLD.BDY.DIST	(real)	result of the computation - distance to nearest boundary

Field Entity Attributes

XC.FLD	(real)	field center coordinates
YC.FLD	(real)	
PXX.FLD	(real)	field boundary coefficients
PYY.FLD	(real)	
PXY.FLD	(real)	

Routines Called

None

Events Scheduled

None

<u>Code</u> See Figure 4

<u>Brief Explanation</u>

Line 9	initializes D to + ∞
Lines 11-14	setup element location and direction
Lines 15-42	loop over each field in FLD.SET
Lines 16-30	compute S1, S2 if they exist, otherwise cycle to next field
Lines 31-38	find minimum positive value among S1, S2, D and save it in D
Line 47	stores the resulting D value (maybe + ∞) in the element's FLD.BDY.DIST attribute

<u>Use</u> The FLD.DIST routine is called for each element in the simulation when the element is created. Whenever an element changes movement direction or crosses a field boundary the routine is called for that element to give a new FLD.BDY.DIST. Note that the distance finally returned is limited to 500m. for reasons of numerical roundoff error. This forces the model to periodically re-evaluate the FLD.BDY.DIST attribute.

```
ROUTINE FLD.DIST
'' COMPUTE DISTANCE FROM VEH TO THE CLOSEST FIELD BOUNDARY
''   ALONG CURRENT DIRECTION OF MOVEMENT
    GIVEN VEH
DEFINE DX,DY,XCUR,YCUR,QX,QY,PXX,PYY,PXY,A,B,C,ARG,SQ,S1,S2,D AS DOUBLE
     VARIABLES
    DEFINE VEH AS AN INTEGER VARIABLE
DEFINE FIELD AS AN INTEGER VARIABLE
    LET D = RINF.C
  IF FLDS.CREATED NE 0
    LET DX = COS.F(DIR.OF.MVMT(VEH))
    LET DY = SIN.F(DIR.OF.MVMT(VEH))
    LET XCUR = X.CURRENT(VEH)
    LET YCUR = Y.CURRENT(VEH)
    FOR EVERY FIELD IN FLD.SET DO
         LET QX = XCUR - XC.FLD(FIELD)
         LET QY = YCUR - YC.FLD(FIELD)
         LET PXX = PXX.FLD(FIELD)
         LET PYY = PYY.FLD(FIELD)
         LET PXY = PXY.FLD(FIELD)
         LET A = PXX*DX**2 + PYY*DY**2 + PXY*DX*DY
         LET B = 2*PXX*QX*DX + 2*PYY*QY*DY + PXY*(QX*DY+QY*DX)
         LET C = PXX*QX**2 + PYY*QY**2 + PXY*QX*QY - 1.0
         LET ARG = B**2 - 4.0*A*C
         IF ARG LE 0
              CYCLE
         ELSE
              LET SQ = SQRT.F(ARG)
              LET S1 = -(B + SQ)/(2.0*A)
              LET S2 = (SQ - B)/(2.0*A)
              IF S1 GT 0.0
                   IF S1 LT D
                        LET D = S1
                   ALWAYS
                   CYCLE
              ELSE
                   IF S2 GT 0.0 AND S2 LT D
                        LET D = S2
                   ALWAYS
              ''ENDIF
         ''ENDIF
    LOOP
    IF D LT RINF.C AND D GT 500.0
         LET D = 500.0
    ALWAYS
  ALWAYS
    LET FLD.BDY.DIST(VEH) = D
    RETURN
END
```

Figure 4. FLD.DIST Routine

III. Interfacing The Field Routines with STAR

A. Overview of Changes Required

Several changes in the existing STAR model code are required to implement the field module. For the most part these changes are simple and fairly obvious, but in one case (the MOVE routine) substantial effort was required. The following existing program segments were changed:

PREAMBLE
MAIN
BL.CREATE
RED.CREATE
MOVE

We detail the required changes in the remaining sections of this chapter. Familiarity with the basic STAR ground model is assumed (refs [1]-[5]).

B. PREAMBLE

1. New global integer variable FLDS.CREATED

2. New system owned set FLD.SET

3. New class of temporary entities FIELD with attributes

   NAM.FLD, XC.FLD, YC.FLD, PXX.FLD, PYY.FLD, PXY.FLD,
   TYP.FLD, P1.FLD, P2.FLD, P3.FLD, P4.FLD
   (see Chapter II for definition and type).
   A FIELD may belong to the FLD.SET.

4. New attributes for the TANK (or UNIT) temporary entity:

FLD.INT.DIST	distance to pending internal action
FLD.BDY.DIST	distance to nearest field boundary
FLD.NO	name of the field involved in any pending internal action
FLD.AKT	code describing the kind of action for pending internal actions

5. FLD.INIT defined as a releasable routine.

C. MAIN

1. Immediately following the call to RES.TERR,

```
CALL    FLD.INIT          RELEASE FLD.INIT
```

to create fields.

D. BL.CREATE

1. Immediately after the call to INIT.POS,

```
CALL    FLD.DIST (TANK)
LET     FLD.INT.DIST(TANK) = RINF.C
```

to initialize field distance attributes.

E. RED.CREATE

1. Immediately after the call to INIT.POS,

```
CALL    FLD.DIST(TANK)
LET     FLD.INT.DIST(TANK)= RINF.C
```

to initialize field distance attributes.

F. MOVE

The changes to the MOVE routine are central to the Field module, and are the most critical. In this section we explain these changes in detail. To aid in understanding the relation of these changes to the rest of the MOVE routine, we include a listing of the entire routine in Figure 5, with changes flagged by "FLD in the right hand margin of the card. The reader is assumed to be familiar with the MOVE routine as documented in reference [4].

1. (Lines 157-161) Whenever a vehicle changes its direction of movement, the previously computed FLD.BDY.DIST is no longer valid, and must be recomputed by a call to FLD.DIST

2. (Lines 170-171) The distance limit for a move increment includes FLD.INT.DIST (to trigger internal actions) and FLD.BDY.DIST + 1 (to trigger boundary actions). The +1 is included to make sure that the boundary is actually crossed (by up to 1 meter) before the action is triggered. This prevents roundoff noise from triggering the same action more than one time.

3. (Lines 215-227) At the end of each movement increment, if there are any fields in the battle, FLD.BDY.DIST and FLD.INT.DIST are updated. If either distance hits zero, the FLD.ACT routine is called to decode and perform the action. If as a result of the field action, the vehicle can no longer move, the call to MOVE is terminated. Finally, if we are within 50 meters of a boundary, the FLD.DIST routine is called to get a more accurate FLD.BDY.DIST. This extra call is required because of round-off errors on the IBM-360 where vehicle X and Y coordinates have about 6 significant digits on a battlefield which may be 100,000 meters deep.

```
ROUTINE TO MOVE GIVEN VEH
DEFINE K AS AN INTEGER VARIABLE
DEFINE SLOPE AS A REAL VARIABLE
DEFINE REM.MOVE.TIME, DEL.X, DEL.Y, D.TO.MCP, ALPH,  SALPH,
    CALPH, GRAD.X, GRAD.Y, SPD.LIMIT, ACCEL.LIMIT, DECEL.LIMIT,
    DIST.LIMIT, DEL.SPD, DIST.INCR, TIME.INCR AS REAL VARIABLES
DEFINE DIST.REQ, TIME.REQ, ZERO.LEVEL AS REAL VARIABLES
DEFINE X.DEST, Y.DEST, DIR, CX, CY, NX, NY, LX, LY, NLX, NLY, PX,PY,
     NPX, NPY, X.OFF, Y.OFF, D.TO.DEST AS REAL VARIABLES
DEFINE THETA, CTH, STH AS REAL VARIABLES
DEFINE VEH, FINAL AS INTEGER VARIABLES
DEFINE MST, RT, NM, MCP.INC, LM, MCP, D.ON.RT AS INTEGER VARIABLES
DEFINE FAKE.MCP AS AN INTEGER VARIABLE
DEFINE I AND J AS INTEGER VARIABLES
CHECKOUT VEH ALWAYS
LET ZERO.LEVEL = 1.0     LET FINAL = 0
LET MST = MV.STATE(VEH)
IF MST = 0 OR MST >= 4 RETURN
ELSE
IF MST EQUALS 1
    CALL RT.SEL(VEH)
    LET MV.STATE(VEH) = 2
ALWAYS
LET REM.MOVE.TIME = TIME.V - T.SPD(VEH)
LET RT = ROUTE(VEH)
LET D.ON.RT = DIR.ON.RT(VEH)
LET FAKE.MCP = 0
IF RT EQUALS 0    LET D.TO.MCP = RINF.C    GO TO ANGLES
ELSE
''CONSISTENCY CHECK FOR POSSIBLE TURNAROUND
IF AREA.START(VEH) LT AREA.END(VEH)
    IF D.ON.RT EQ 0  GO TO NEW.MCP
    ELSE LET D.ON.RT = 0
    LET DIR.ON.RT(VEH) = 0
    LET K = DIM.F(ROUTE.DATA(RT,*))/3
    LET MCP = NEXT.MCP(VEH)
    IF MCP EQ 0 LET NEXT.MCP(VEH) = 1
    ELSE IF MCP EQ K  LET NEXT.MCP(VEH) = 0
        ELSE LET NEXT.MCP(VEH) = MCP + 1
        ALWAYS
    ALWAYS
ELSE ''AREA.START GT AREA.END
    IF D.ON.RT EQ 1 GO TO NEW.MCP
    ELSE LET D.ON.RT = 1
    LET DIR.ON.RT(VEH) = 1
    LET K = DIM.F(ROUTE.DATA(RT,*))/3
    LET MCP = NEXT.MCP(VEH)
    IF MCP EQ 0 LET NEXT.MCP(VEH) = K
    ELSE IF MCP EQ 1 LET NEXT.MCP(VEH) = 0
        ELSE LET NEXT.MCP(VEH) = MCP -1
```

Figure 5. MOVE Routine

```
            ALWAYS
        ALWAYS
    ALWAYS
    'NEW.MCP' LET MCP = NEXT.MCP(VEH)     LET NM = MCP * 3
    IF MCP EQUALS 0  ''MOVE TO POSITION IN AREA.END
        IF POS.IN.PLT.AREA(VEH) EQUALS 0, CALL BEST.POS(VEH)   ''SETTING POS.IN.PLT.A
        ALWAYS
        LET I = PLT(VEH)      LET K = POS.IN.PLT.AREA(VEH) * 3
        FOR J = 1 TO DIM.F(POSITION(I,*,*)) WITH POSITION(I,J,1) EQUALS
             AREA.END(VEH)
        DO
             LET X.DEST = POSITION(I,J,K-1)
             LET Y.DEST = POSITION(I,J,K)
             LET DIR = POSITION(I,J,K+1)
        LOOP
        LET D.TO.MCP = SQRT.F((X.DEST-X.CURRENT(VEH))**2 +
             (Y.DEST-Y.CURRENT(VEH))**2)
        IF D.TO.MCP LESS THAN ZERO.LEVEL,
             LET MV.STATE(VEH) = 4
             LET DIR.OF.MVMT(VEH) = DIR
             LET PRI.DIR(VEH) = DIR
             LET SPD(VEH) = 0.
             LET FINAL = 1
             GO TO NEW.INCR
        ELSE
        GO TO DIRN.COMP
    ELSE
    IF FORM.CODE(VEH) EQUALS 0       ''GO DIRECTLY TO NEXT MCP
        LET X.DEST = ROUTE.DATA(RT,NM-2)
        LET Y.DEST = ROUTE.DATA(RT,NM-1)
        LET D.TO.MCP = SQRT.F((X.DEST-X.CURRENT(VEH))**2 +
             (Y.DEST-Y.CURRENT(VEH))**2)
        GO TO DIRN.COMP
    ELSE     ''MOVE ALONG ROUTE OFFSET BY FORMATION
    IF MCP EQUALS 1 AND D.ON.RT EQUALS 0  ''TOWARD FIRST MCP
             LET NX = ROUTE.DATA(RT,4)
             LET NY = ROUTE.DATA(RT,5)
             LET LX = ROUTE.DATA(RT,1)
             LET LY = ROUTE.DATA(RT,2)
             LET I = ROUTE.DATA(RT,3)
    ELSE
        LET K = DIM.F(ROUTE.DATA(RT,*))
        IF MCP EQUALS K/3 AND D.ON.RT EQUALS 1   ''TOWARD LAST MCP GOING BACKWARD
             LET NX = ROUTE.DATA(RT,K-5)
             LET NY = ROUTE.DATA(RT,K-4)
             LET LX = ROUTE.DATA(RT,K-2)
             LET LY = ROUTE.DATA(RT,K-1)
             LET I = ROUTE.DATA(RT,K-3)
        ELSE GO TO INTERMED
        ALWAYS
```

Figure 5 (Continued).

```
ALWAYS
    LET NLX = NX-LX       LET NLY = NY-LY
    IF I EQUALS 0,  LET I = FORM.CODE(VEH)
    ALWAYS
    LET J =    FORM.POS(VEH) * 2
    LET X.OFF = FORM.OFFSET(I,J-1)
    LET Y.OFF = FORM.OFFSET(I,J)
    LET THETA = ARCTAN.F(NLY,NLX)
    LET CTH = COS.F(THETA)
    LET STH = SIN.F(THETA)
    LET X.DEST = LX + (X.OFF + FOR.CHG.INT)*CTH  - Y.OFF*STH
    LET Y.DEST = LY + (X.OFF + FOR.CHG.INT)*STH + Y.OFF * CTH
    LET D.TO.MCP = SQRT.F((X.DEST-X.CURRENT(VEH))**2 +(Y.DEST-Y.CURRENT(VEH))
         **2)
    GO TO DIRN.COMP
'INTERMED'      ''TO HERE FOR INTERMEDIATE MCP'S ON ROUTE
    LET CX = X.CURRENT(VEH)      LET CY = Y.CURRENT(VEH)
    IF D.ON.RT EQUALS 0  LET LM = NM - 3
    ELSE LET LM = NM + 3
    ALWAYS
    LET NX = ROUTE.DATA(RT,NM-2)
    LET NY = ROUTE.DATA(RT,NM-1)
    LET LX = ROUTE.DATA(RT,LM-2)
    LET LY = ROUTE.DATA(RT,LM-1)
    LET NLX = NX - LX          LET NLY = NY - LY
    LET ALPH =-((CX-NX)*NLX + (CY-NY)*NLY) / (NLX*NLX + NLY*NLY)
    LET PX = ALPH * LX + (1. - ALPH) * NX
    LET PY = ALPH * LY + (1. - ALPH) * NY
    LET NPX = NX - PX          LET NPY = NY - PY
    LET I = ROUTE.DATA(RT,NM+3*(D.ON.RT-1))
    IF I EQUALS 0,  LET I = FORM.CODE(VEH)
    ALWAYS
    LET J =    FORM.POS(VEH) * 2
    LET X.OFF = FORM.OFFSET(I,J-1)
    LET Y.OFF = FORM.OFFSET(I,J)
    LET D.TO.MCP = SQRT.F(NPX*NPX + NPY*NPY)
    IF D.TO.MCP LESS THAN ZERO.LEVEL   GO TO MCP.REACHED
    ELSE
    LET THETA = ARCTAN.F(NLY,NLX)
    LET CTH = COS.F(THETA)
    LET STH = SIN.F(THETA)
    LET ALPH = FOR.CHG.INT / D.TO.MCP
    LET X.DEST = PX + ALPH * NPX + X.OFF  * CTH - Y.OFF * STH
    LET Y.DEST = PY + ALPH * NPY + Y.OFF * CTH + X.OFF * STH
    LET D.TO.DEST = SQRT.F((X.DEST-CX)**2 + (Y.DEST -CY)**2)
    IF D.TO.DEST IS LESS THAN D.TO.MCP
         LET D.TO.MCP = D.TO.DEST
         LET FAKE.MCP = 1
    ELSE LET FAKE.MCP = 0
    ALWAYS
```

Figure 5 (Continued).

```
'DIRN.COMP'
IF D.TO.MCP IS LESS THAN ZERO.LEVEL,
    GO TO MCP.REACHED
ELSE
LET DEL.X = X.DEST - X.CURRENT(VEH)
LET DEL.Y = Y.DEST - Y.CURRENT(VEH)
LET DIR = DIR.OF.MVMT(VEH)                                                ''FLD
LET DIR.OF.MVMT(VEH) = ARCTAN.F(DEL.Y,DEL.X)
IF ABS.F(DIR-DIR.OF.MVMT(VEH)) GT 0.02                                    ''FLD
    CALL FLD.DIST(VEH)           ''TO COMPUTE NEW FLD.BDY.DIST             FLD
ALWAYS                                                                    ''FLD
'ANGLES'
LET SALPH = SIN.F(DIR.OF.MVMT(VEH))      LET CALPH = COS.F(DIR.OF.MVMT(VEH))
'NEW.INCR' CALL ELEVG(X.CURRENT(VEH),Y.CURRENT(VEH)) YIELDING Z.CURRENT(VEH),
GRAD.X, GRAD.Y
IF FINAL EQUALS 1, GO TO END.MOVE
ELSE
LET SLOPE = GRAD.X * CALPH + GRAD.Y * SALPH
CALL MOVE.LIMITS GIVEN VEH, SLOPE YIELDING SPD.LIMIT, ACCEL.LIMIT, DECEL.LIMIT
LET DIST.LIMIT = MIN.F(D.TO.MCP,MAX.DIST.INCR,                           ''FLD
     FLD.INT.DIST(VEH), (1 + FLD.BDY.DIST(VEH)))                          ''FLD
LET DEL.SPD= SPD.LIMIT - SPD(VEH)
IF ABS.F(DEL.SPD) LT 0.1,
''EASY CASE -- NO ACCELERATION --
    LET DIST.INCR = REM.MOVE.TIME * SPD.LIMIT
    IF DIST.INCR IS GREATER THAN DIST.LIMIT,
    ''MOVE STOPPED BY DISTANCE LIMIT
         LET DIST.INCR = DIST.LIMIT
         LET TIME.INCR = DIST.INCR / SPD.LIMIT
    ELSE
    ''MOVE STOPPED BY TIME LIMIT
         LET TIME.INCR = REM.MOVE.TIME
    ALWAYS GO TO MOVE.IT
ELSE  ''HARD CASE -- ACCELERATION REQUIRED --
    IF DEL.SPD IS LESS THAN 0, LET ACCEL.LIMIT = DECEL.LIMIT
    ALWAYS LET TIME.REQ = DEL.SPD / ACCEL.LIMIT
    LET DIST.REQ = SPD(VEH)*TIME.REQ + 0.5 * ACCEL.LIMIT * TIME.REQ **2
    IF TIME.REQ IS GREATER THAN REM.MOVE.TIME,
    ''SPD.LIMIT CANNOTBE ATTAINED IN REMAINING TIME, SO CHANGE LIMIT
         LET SPD.LIMIT = SPD(VEH) + ACCEL.LIMIT * REM.MOVE.TIME
         LET DIST.INCR = SPD(VEH) * REM.MOVE.TIME + 0.5 * ACCEL.LIMIT *
              REM.MOVE.TIME ** 2
    ELSE   ''SPD.LIMIT CAN BE ATTAINED
         LET DIST.INCR = DIST.REQ + (REM.MOVE.TIME - TIME.REQ)*SPD.LIMIT
    ALWAYS
    IF DIST.INCR IS LESS THAN DIST.LIMIT,
    ''MOVE WILL BE STOPPED BY TIME LIMIT
         LET TIME.INCR = REM.MOVE.TIME
         LET SPD(VEH) = SPD.LIMIT
    ELSE    ''MOVE STOPPED BY DISTANCE LIMIT
```

Figure 5 (Continued).

```
          LET DIST.INCR = DIST.LIMIT
         IF DIST.LIMIT IS LESS THAN DIST.REQ,
              LET TIME.INCR = (SQRT.F(SPD(VEH)**2+2.*ACCEL.LIMIT*DIST.LIMIT)
                   -SPD(VEH))/ACCEL.LIMIT
              ADD TIME.INCR * ACCEL.LIMIT TO SPD(VEH)
         ELSE
              LET TIME.INCR = TIME.REQ +(DIST.LIMIT-DIST.REQ)/SPD.LIMIT
              LET SPD(VEH)=SPD.LIMIT
         ALWAYS
    ALWAYS
'MOVE.IT'     SUBTRACT TIME.INCR FROM REM.MOVE.TIME
ADD DIST.INCR * CALPH TO X.CURRENT(VEH)
ADD DIST.INCR * SALPH TO Y.CURRENT(VEH)
SUBTRACT DIST.INCR FROM D.TO.MCP
IF FLDS.CREATED GT 0
SUBTRACT DIST.INCR FROM FLD.BDY.DIST(VEH)                                    ''FLD
SUBTRACT DIST.INCR FROM FLD.INT.DIST(VEH)                                    ''FLD
IF FLD.INT.DIST(VEH) LE 0.0 OR FLD.BDY.DIST(VEH) LE 0.0                      ''FLD
    CALL FLD.ACT(VEH)                                                        ''FLD
     IF KKILL(VEH) EQ 1 OR MKILL(VEH) EQ 1                                   ''FLD
          LET FINAL = 1                                                      ''FLD
     ALWAYS                                                                  ''FLD
ALWAYS                                                                       ''FLD
IF FLD.BDY.DIST(VEH) LT 50.0
    CALL FLD.DIST(VEH)
ALWAYS
ALWAYS
IF REM.MOVE.TIME LT 0.01, LET FINAL=1
REGARDLESS
IF D.TO.MCP IS LESS THAN ZERO.LEVEL,
'MCP.REACHED'
    IF FAKE.MCP EQUALS 1
         LET FAKE.MCP = 0
         GO TO NEW.MCP
    ELSE
    IF MCP = 0
         LET FINAL = 1
         GO TO NEW.MCP
    ELSE
    IF DIR.ON.RT(VEH) EQUALS 0    ''MCP NUMBERS INCREASE
         IF NEXT.MCP(VEH) EQUALS DIM.F(ROUTE.DATA(ROUTE(VEH),*))/3
              LET NEXT.MCP(VEH) = 0
              GO TO NEW.MCP
         ELSE
              ADD 1 TO NEXT.MCP(VEH)      GO TO NEW.MCP
    ELSE      ''MCP NUMBERS DECREASE
         IF NEXT.MCP(VEH) EQUALS 1
              LET NEXT.MCP(VEH)=0
              GO TO NEW.MCP
         ELSE
```

Figure 5 (Continued).

```
                SUBTRACT 1 FROM NEXT.MCP(VEH)    GO TO NEW.MCP
ELSE  GO TO NEW.INCR
'END.MOVE'     LET T.SPD(VEH) = TIME.V
RETURN
END
```

Figure 5 (Continued).

IV. Field Actions

The final aspect of the Field module which we must discuss is the implementation of field actions. As indicated in Chapter I, field actions occur as a result of a vehicle encountering a field boundary. Actions which occur immediately are called field boundary actions. Actions which are planned for later occurrence are called field internal actions.

A. Deciding Which Action to Perform

When the FLD.ACT routine is called from the MOVE routine, it must decide which field action(s) to perform. Figure 6 shows a partial listing of a FLD.ACT routine. Details of particular actions are not included since they are quite dependent on the scenario plan for a given study. Examples of how typical actions might be handled are given in section B of this chapter. What this code segment does show is the procedure for deciding whether entry,internal, or exit actions should occur, and in each case, the field involved.

If FLD.INT.ACT(VEH) has been decremented to zero, then a previously planned internal action should occur, (lines 4-11). The field for which this action was planned is indicated in the vehicle's FLD.NO attribute and the action type is given by the vehicle's FLD.AKT attribute. Note that at most one such action can be pending at any one time. A typical internal action is a mine detonation which would call POP.A.MINE to assess lethality and, if the vehicle survived, would initiate evasive action and plan the next detonation for this vehicle.

If FLD.BDY.DIST(VEH) has been decremented to zero, then a boundary action should occur. A given move increment may, in some circumstances, cross more than one field boundary. Thus, instead of storing the field

```
ROUTINE FLD.ACT(VEH)    '' SORTS OUT WHICH FIELD ACTIONS TO PERFORM
DEFINE VEH,FIELD AS INTEGER VARIABLES
DEFINE DX,DY,XCUR,YCUR,QX,QY,PXX,PXY,PYY,A,B,C,ARG,SQ,S1,S2 AS DOUBLE VARIABLES
IF FLD.INT.DIST(VEH)  LE 0.0
     PRINT 1 LINE WITH NAME(VEH),X.CURRENT(VEH),Y.CURRENT(VEH),FLD.NO(VEH),
          TIME.V AS FOLLOWS
FLD INT ACT  VEH=****  LOCN = ****** ******  FIELD = ****  TIME = ****
''
'' HERE WE CALL ROUTINES TO PERFORM INTERNAL ACTIONS
''
ALWAYS
IF FLD.BDY.DIST(VEH) LE 0.0
    LET DX = COS.F(DIR.OF.MVMT(VEH))
    LET DY = SIN.F(DIR.OF.MVMT(VEH))
    LET XCUR = X.CURRENT(VEH)
    LET YCUR = Y.CURRENT(VEH)
    FOR EVERY FIELD IN FLD.SET DO
         LET QX = XCUR - XC.FLD(FIELD)
         LET QY = YCUR - YC.FLD(FIELD)
         LET PXX = PXX.FLD(FIELD)
         LET PYY = PYY.FLD(FIELD)
         LET PXY = PXY.FLD(FIELD)
         LET A = PXX*DX**2 + PYY*DY**2 + PXY*DX*DY
         LET B = 2*PXX*QX*DX + 2*PYY*QY*DY + PXY*(QX*DY+QY*DX)
         LET C = PXX*QX**2 + PYY*QY**2 + PXY*QX*QY - 1.0
         LET ARG = B**2 - 4.0*A*C
         IF ARG LE 0
              CYCLE
         ELSE
              LET SQ = SQRT.F(ARG)
              LET S1 = -(B + SQ)/(2.0*A)
              LET S2 = (SQ - B)/(2.0*A)
              IF S1 LE 0.0 AND S1 GE -2.0
                   PRINT 1 LINE WITH NAME(VEH),XCUR,YCUR,NAM.FLD(FIELD),
                        TIME.V AS FOLLOWS
FIELD ENTRY  VEH=**** LOCN = ****** ******  FIELD = ****  TIME = ****
''
''  HERE WE CALL ROUTINES TO PERFORM FIELD ENTRY BOUNDARY ACTIONS
''
              ALWAYS
              IF S2 LE 0.0 AND S2 GE -2.0
                   PRINT 1 LINE WITH NAME(VEH),XCUR,YCUR,NAM.FLD(FIELD),
                        TIME.V AS FOLLOWS
FIELD EXIT   VEH=**** LOCN = ****** ******  FIELD = ****  TIME = ****
''
''  HERE WE CALL ROUTINES TO PERFORM FIELD EXIT BOUNDARY ACTIONS
''
                   IF NAM.FLD(FIELD) EQ FLD.NO(VEH)
                        LET FLD.INT.DIST(VEH) = RINF.C
                        LET FLD.NO(VEH) = 0
```

Figure 6. Partial FLD,ACT Routine

```
                    ALWAYS
                ALWAYS
            ''ENDIF
        LOOP
    ALWAYS
    CALL FLD.DIST(VEH)
    RETURN
    END
```

Figure 6 (Continued).

number for a boundary action on the vehicle, we test each field for boundary crossing whenever FLD.ACT is called with FLD.BDY.DIST $\leq$ 0. The loop to implement this test is contained in lines 17 to 54. Note the tolerance of $\pm$ 1 meter in lines 33 and 41. If this tolerance is decreased, some field entries or exits may be missed.

Regardless of the actions triggered in FLD.ACT, the FLD.BDY.DIST is recomputed prior to return to set up the next boundary crossing.

B. Implementing Some Typical Actions

In this section we provide suggestions for implementing some possible field actions. In many cases, an action performed on entering a field must be paired with a complementary action on exit which restores the vehicle to its previous state. This requires either saving the previous state for those attributes affected by the field or having a default background state to which exiting vehicles return.

The choice of field action to perform may, in some cases, depend on the type of "vehicle" encountering the field - an obstacle that totally stops wheeled vehicles may merely slow tanks and may be no hindrance at all for dismounted infantry. In this section we assume that the <u>choice</u> of actions to perform will be determined by the scenario writers and tacticians.

1. Change Speed

A frequent result of entering a field is a change in speed (e.g. for an obstacle field). In this section we discuss speed changes which do not completely stop the vehicle. Simply changing the vehicle's SPD attribute on field entry will <u>NOT</u> suffice, because this attribute is reset at every move increment based on the results of a call to routine MOVE.LIMITS. One way of implementing a speed change would be to define a new vehicle

attribute FLD.SPD.FAC the field speed factor for the vehicle. When the vehicle is not in any field, FLD.SPD.FAC assumes a background value of 1.0. On field entry the FLD.ACT routine would set FLD.SPD.FAC to a value greater than 1.0 for a speed increase and less than 1.0 for a speed decrease. (The value would be one of the field parameters P1 - P4). The MOVE.LIMITS routine would include FLD.SPD.FAC in its computation of vehicle target speed for each move increment. On field exit, FLD.ACT sets FLD.SPD.FAC back to 1.0.

2. Stop for T Seconds

To simulate a brief delay on field entry, the FLD.ACT routine can simply set the vehicle's MV.STATE to 3 (Stop along route). The vehicle will stop as soon as it is physically possible. For a longer delay, we probably also want to set the vehicle's DEFNUM to simulate taking advantage of available cover. In either case, once the vehicle is stopped it will not trigger further field action. To get the vehicle started again after T seconds (a field parameter), a new event must be scheduled by the field entry routine. No field action is required on field exit. Similar comments apply to the hasty defense action.

3. Response to Minefields

A variety of actions may be appropriate for simulating a vehicle encountering a minefield. The first question which must be asked is whether the vehicle notices the minefield: Several cases can occur:

a) Minefield detected on entry - Then the field entry action routine can initiate any desired vehicle responses and will plan an internal mine detonation action.

b) Minefield not detected - Then the field entry action routine only plans the internal mine detonation action, and the vehicle simply drives on. When the first detonation occurs, the minefield is assumed detected and appropriate responses are taken by the detonating vehicle and other vehicles in his immediate unit.

c) Minefield detected prior to entry - A clearly visible field can be modelled by two concentric field ellipses. Entering the larger ellipse triggers the field detection and any responses thereto. Entering the smaller ellipse initiates mine detonation actions.

Given that a vehicle has detected the minefield, several actions may be appropriate for it and for other vehicles in its immediate unit (perhaps platoon). If mine plows or rollers are available, they may be used (vehicle PLOW.COND attribute). This probably requires a brief stop. Vehicles without plows may line up behind those with plows (a formation change). The resulting unit will move slower while plowing (speed change).

A more difficult action is an attempt to bypass the field. This would involve creation of special routes for moving around the field and has not been worked out in detail.

On field exit, if we assume that the vehicle(s) can detect the exit, the above actions can be cancelled (perhaps involving another brief delay). If the exit is not detectable, then concentric ellipses could be used to delay the change back to a normal condition until after the field has been left.

4. Mine Detonation

When a vehicle enters a minefield, the field boundary action is responsible for planning a mine detonation and storing this as the internal action for the vehicle. The distance that the vehicle will move before detonating a mine can be determined using a random draw from a distribution whose parameters are influenced by

a) minefield density

b) vehicle geometry (e.g. track width)

c) mine detectability

d) whether vehicle is in a cleared path (e.g. following a vehicle with a plow)

This distance is used for the vehicle's FLD.INT.DIST attribute. If the vehicle travels this far through the field before encountering the exit boundary, then a field internal action is triggered. This action should assess the lethality of the mine, and, if the vehicle can still move, should plan the next detonation.

As discussed in 3) above, if the minefield is not detectable, then the first detonation in a platoon or company might trigger responses from all vehicles in that unit.

5. Call for Artillery or Air Strike

One possible use for field ellipses is to represent pre-planned artillery or close air support trigger areas. The presence of a threshold number of enemy vehicles in the area can be used to initiate requests for artillery or air missions. Implementing this concept using fields calls for establishing a counter for each such field to keep track of the number of enemy inside the field. On field entry, the counter is incremented, and perhaps a mission request is initiated. On field exit the counter is decremented.

Other possible uses for the Field module will no doubt become apparent in the future. Because of the great variety of possible combinations of the various actions, actual implementation of the actions must await guidance.

REFERENCES

[1] HAGEWOOD, E. G. and WALLACE, W.S., Simulation of Tactical Alternative Responses (STAR), M.S. Thesis, Naval Postgraduate School, December 1978.

[2] HARTMAN, J.K., "Parametric Terrain and Line of Sight Modelling in the STAR Combat Model", Naval Postgraduate School, Technical Report NPS55-79-018, August 1979.

[3] CALDWELL, W.J. and MEIERS, W.D., An Air to Ground and Ground to Air Combined Arms Combat Simulation (STAR-AIR), M.S. Thesis, Naval Postgraduate School, September 1979.

[4] PARRY, S.H. and KELLEHER, E.P. Jr., "Tactical Parameters and Input Requirements for the Ground Component of the STAR Combat Model," Naval Postgraduate School, Technical Report NPS55-79-023, October 1979.

[5] HARTMAN, J. K., "Ground Movement Modelling in the STAR Combat Model," Naval Postgraduate School, Technical Report NPS 55-80-021, May 1980.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
4.	Professor James K. Hartman, Code 55Hh Department of Operations Research Naval Postgraduate School Monterey, California 93940	40
5.	Professor S. H. Parry, Code 55Py Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
6.	LTC Edward P. Kelleher, Code 55Ka Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
7.	Professor Arthur L. Schoenstadt, Code 53Zh Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
8.	Office of the Commanding General U.S. Army TRADOC ATTN: General Donn A. Starry Ft. Monroe, Virginia 23651	1
9.	Headquarters U.S. Army TRADOC ATTN: ATCG-T (Colonel Ed Scribner) Ft. Monroe, Virginia 23651	1
10.	Headquarters U.S. Army TRADOC ATTN: Director, Analysis Directorate Combat Developments Ft. Monroe, Virginia 23651	1
11.	Headquarters U.S. Army TRADOC ATTN: Director, Maneuver Directorate Combat Developments (COL Fred Franks) Ft. Monroe, Virginia 23651	1

12. Mr. David Hardison 1
Deputy Under Secretary of the Army
(Operations Research)
Department of the Army, The Pentagon
Washington, D.C. 20310

13. LTG William Richardson 1
Commanding General
U.S. Army Combined Arms Center
Ft. Leavenworth, Kansas 66027

14. Director 1
Combined Arms Combat Development Activity
ATTN: COL Reed
Ft. Leavenworth, Kansas 66027

15. Director, BSSD 1
Combined Arms Training Development Activity
ATTN: ATZLCA-DS
Fort Leavenworth, Kansas 66027

16. Director 1
Combat Analysis Office
ATTN: Mr. Kent Pickett
U.S. Army Combined Arms Center
Fort Leavenworth, Kansas 66027

17. Command and General Staff College 1
ATTN: Education Advisor
Room 123, Bell Hall
Fort Leavenworth, Kansas 66027

18. Dr. Wilbur Payne 1
Director
U.S. Army TRADOC Systems Analysis Activity
White Sands Missile Range, New Mexico 88002

19. Headquarters 1
Department of the Army
Office of the Deputy Chief of Staff
for Operations and Plans
ATTN: LTG Glenn Otis
Washington, D.C. 20310

20. Commander 1
U.S. Army Concepts Analysis Agency
8120 Woodmont Avenue
ATTN: MOCA-SMS (CPT Steve Shupack)
Bethesda, Maryland 20014

21. Commander 1
U.S. Army Concepts Analysis Agency
ATTN: LTC Earl Darden-MOCA-WG
8120 Woodmont Avenue
Bethesda, Maryland 20014

22. Director 1
U.S. Army Night Vision & Electro-optical Lab.
ATTN: DEL-NV-V1 (Mr. Bob Hermes)
Fort Belvoir, VA 22060

23. Director 1
U.S. Army Material Systems Analysis Activity
ATTN: Mr. Will Brooks
Aberdeen Proving Grounds, Maryland 21005

24. Director 1
Armored Combat Vehicle Technology Program
ATTN: COL Fitzmorris
U.S. Army Armor Center
Fort Knox, Kentucky 40121

25. Colonel Frank Day 1
TRADOC Systems Manager-XM1
U.S. Army Armor Center
Fort Knox, Kentucky 40121

26. Director 1
Combat Developments, Studies Division
ATTN: MAJ W. Scott Wallace
U.S. Army Armor Agency
Fort Knox, KY 40121

27. Commandant 1
U.S. Army Field Artillery School
ATTN: ATSF-MBT (CPT Steve Starner)
Fort Sill, Oklahoma 73503

28. Director 1
Combat Developments
ATTN: COL Clark Burnett
U.S. Army Aviation Agency
Fort Rucker, Alabama 36360

29. Director 1
Combat Developments
U.S. Army Infantry Agency
Fort Benning, GA 31905

30. Director 1
Missile Intelligence Agency
ATTN: ADA Tactics (CPT E.G. Hagewood)
Redstone Arsenal, AL 35809

31. Director 1
Combat Developments
ATTN: CPT William D. Meiers
U.S. Army Air Defense Agency
Fort Bliss, TX 79905

32. Commander 1
U.S. Army Logistics Center
ATTN: ATCL-OS-Mr. Cammeron/CPT Schuessler
Fort Lee, VA 23801

33.	Commander USAMMCS ATTN: ATSK-CD-CS-Mr. Lee/Mr. Marmon Redstone Arsenal, AL 35809	1
34.	Commander U.S. Army Combined Arms Center ATTN: ATZL-CA-CAT (R.E. DeKinder, Jr.) Fort Leavenworth, KA 66027	1
35.	Director U.S. Army AMSAA ATTN: DRXSY-AA (Mr. Tom Coyle) Aberdeen Proving Grounds, MD 21005	1
36.	Chief TRADOC Research Element Monterey (TREM) Naval Postgraduate School Monterey, CA 93940	10
37.	Office of the Deputy Chief of Staff for Combat Developments U.S. Army TRADOC ATTN: Major General Carl Vuono Fort Monroe, Virginia 23651	1
38.	Deputy Commanding General Combined Arms Combat Development Activity ATTN: ATZL-CA-DC (BG(P) Jack Walker) Fort Leavenworth, KA 66027	1
39.	Commanding General U.S. Army Infantry Center ATTN: Major General David Grange Fort Benning, GA 31905	1
40.	Director Combat Developments ATTN: COL Pokorney U.S. Army Field Artillery Center Fort Sill, Oklahoma 73503	1
41.	Director Combat Developments ATTN: COL Gus Watt U.S. Army Infantry School Fort Benning, GA 31905	1
42.	Director USATRASANA ATTN: Mr. Ray Heath White Sands Missile Range, New Mexico 88002	1
43.	R. Stampfel, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1